

Choice Lists

Suppose that the properties of `<fu>` are listed in a choice complex-type in the target schema. Assume again, as above, that `fu` is mapped to an ontological class `Foo`, with each of `bari` mapped to a property, `Foo.bari`. Assume further, as above, that the source XML schema has an Xpath pattern `foo` that maps to the ontological class `Foo`, with further children patterns `foo/barr1`, `foo/barr2`, etc., mapping to the relevant property paths.

Preferably, the general rule for producing XSLT script associated with a target choice bloc is as follows. Start with the tags `<xsl:choose>` `</xsl:choose>`. For each element in the choice sequence, insert into the choose bloc `<xsl:when test="barr">` `</xsl:when>` and within that bloc insert code appropriate to the cardinality restrictions of that element, exactly as above for sequence blocs, including the creation of new templates if needed. Finally, if there are no elements with `minOccurs="0"` in the choice bloc, select any tag `<barr>` at random in the choice bloc, and insert into the XSLT, right before the closing `</xsl:choose>`, `<xsl:otherwise>` `<barr>` `</barr>` `</xsl:otherwise>`.

As an exemplary illustration, suppose the complexType appears in the target schema as follows:

```
<xsl:choose>
  <xsl:element name="bar1" type="xs:string" />
  <xsl:element name="bar2" type="xs:string" minOccurs="0" maxOccurs="7"/>
  <xsl:element name="bar3" type="xs:string" minOccurs="1" maxOccurs="8"/>
  <xsl:element name="bar4" type="xs:string" minOccurs="3" maxOccurs="unbounded"/>
  <xsl:element name="bar5" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>

  <xsl:element name="barr" type="xs:string" />
</xsl:choose>
```

Then, based on the above cases, the following XSLT script is generated.

```
<fu>
  <xsl:choose>
    <xsl:when test="bar1">
      <barr1>
        <xsl:value-of select="bar1"/>
      </barr1>
    </xsl:when>
    <xsl:when test="bar2">
      <xsl:for-each select="bar2[position() <= 8]">
        <barr2>
          <xsl:value-of select="."/>
        </barr2>
      </xsl:for-each>
    </xsl:when>
    <xsl:when test="bar3">
      <xsl:for-each select="bar3[position() <= 9]">
        <barr3>
```

```

        <xsl:value-of select="."/>
    </barr3>
</xsl:for-each>
<xsl:call-template name="generate_barr3">
    <xsl:with-param name="so_far" select="count(bar3)"/>
</xsl:call-template>
</xsl:when>
<xsl:when test="bar4">
    <xsl:for-each select="bar4">
        <barr4>
            <xsl:value-of select="."/>
        </barr4>
    </xsl:for-each>
    <xsl:call-template name="generate_barr4">
        <xsl:with-param name="so_far" select="count(bar4)"/>
    </xsl:call-template>
</xsl:when>
<xsl:when test="bar5">
    <xsl:for-each select="bar5">
        <barr5>
            <xsl:value-of select="."/>
        </barr5>
    </xsl:for-each>
</xsl:when>
    <xsl:otherwise>
    </xsl:otherwise>
</xsl:choose>
</fu>
</xsl:template>

<xsl:template match="text()|@*"/>

<xsl:template name="generate_barr3">
    <xsl:param name="so_far"/>
    <xsl:if test="$so_far < 1">
        <barr3>
        </barr3>
        <xsl:call-template name="generate_barr3">
            <xsl:with-param name="so_far" select="$so_far + 1"/>
        </xsl:call-template>
    </xsl:if>
</xsl:template>

<xsl:template name="generate_barr4">
    <xsl:param name="so_far"/>
    <xsl:if test="$so_far < 3">
        <barr4>
        </barr4>
        <xsl:call-template name="generate_barr4">
            <xsl:with-param name="so_far" select="$so_far + 1"/>
        </xsl:call-template>
    </xsl:if>
</xsl:template>

```

All Lists

Suppose that the properties of `<fu>` are listed in an all complex-type in the target schema. Assume again, as above, that `foo` is mapped to an ontological class `Foo`, with each of `bari` mapped to a property, `Foo.bari`. Assume further that the source XML schema has an Xpath pattern `foo` that maps to the ontological class `Foo`, with further children patterns `foo/barr1`, `foo/barr2`, etc., mapping to the relevant property paths.

In a preferred embodiment of the present invention, a general rule is to test for the presence of each of the source tags associated with the target tags, by way of

```
<xsl:if test="foo">
  <fu>
    <xsl:value-of select="foo"/>
  </fu>
</xsl:if>
```

Preferably, if any of the elements has `minOccurs="1"` then the negative test takes place as well:

```
<xsl:if test="not (foo)">
  <fu>
    </fu>
</xsl:if>
```

As an exemplary illustration, suppose the complexType appears in the target schema as follows:

```
<xs:complexType name="bar">
  <xs:all>
    <xs:element name="bar2" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="bar3" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:all>
</xs:complexType>
```

Then the following XSLT script is generated.

```
<fu>
<xsl:template match="foo">
  <xsl:if test="position() = 1">
    <xsl:if test="bar1">
      <barr1>
        <xsl:value-of select="bar1"/>
      </barr1>
    </xsl:if>
    <xsl:if test="bar2">
      <barr2>
        <xsl:value-of select="bar2"/>
      </barr2>
    </xsl:if>
    <xsl:if test="not (bar2)">
      <barr2>
      </barr2>
    </xsl:if>
  </xsl:if>
</xsl:template>
```

6. In a preferred embodiment of the present invention, when the elements of foo/bar1, foo/bar2, etc. have been processed as above in step 5, everything repeats in a recursive manner for properties that are related to each of the bar_i elements. That is, if the target XML schema has further tags that are children of bar1, bar2, etc., then preferably each of those is treated as properties of the respective target classes of bar1, bar2, and so on, and the above rules apply recursively.

Additional Considerations

In reading the above description, persons skilled in the art will realize that there are many apparent variations that can be applied to the methods and systems described. A first variation to which the present invention applies is a setup where source relational database tables reside in more than one database. The present invention preferably operates by using Oracle's cross-database join, if the source databases are Oracle databases. In an alternative embodiment, the present invention can be applied to generate a first SQL query for a first source database, and use the result to generate a second SQL query for a second source database. The two queries taken together can feed a target database.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made to the specific

exemplary embodiments without departing from the broader spirit and scope of the invention as set forth in the appended claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

10053045-011502